Oracle R Advanced Analytics for Hadoop 2.7.1 Release Notes

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction

Oracle R Advanced Analytics for Hadoop (ORAAH) is a collection of R packages that enable big data analytics from the R environment. With ORAAH, Data Scientists and Data Analysts who are comfortable within R gain the benefit of a broad reach from the R environment into data resident across multiple platforms (HDFS, Hive, Oracle Database, and local files). This gain is boosted by the power to leverage the massively distributed Hadoop computational infrastructure to analyze potentially rich cross-platform datasets.

ORAAH provides some big advantages for big data analytics:

1. A general computation framework where you can use the R language to write custom logic as mappers or reducers. These execute in a distributed, parallel manner using the compute and storage resources available within a Hadoop cluster. Support for binary RData representation of input data enables R-based MapReduce jobs to match the I/O performance of pure Java-based MapReduce programs.
2. Tools ready for work "right out of the box" to provide predictive analytic techniques for linear regression, generalized linear models, neural networks, matrix completion using low rank matrix factorization, non-negative matrix factorization, k-means clustering, principal components analysis and multivariate analysis. While all of these techniques have R interfaces, they have been implemented either in Java, or, in R as distributed, parallel MapReduce jobs that leverage all the nodes of your Hadoop cluster. This enables these analytics tools to work with big data.
3. Support for Apache Spark MLlib functions, Oracle Formula support, and Distributed Model Matrix data structure. New predictive analytic functions from MLlib are exported as R functions and are fully integrated with ORAAH platform. MLlib R functions can be executed either on a Hadoop cluster using YARN to dynamically form a Spark cluster, or on a dedicated standalone Spark cluster. One can switch Spark execution on or off with the new spark.connect() and spark.disconnect() functions. For more information about Apache Spark refer to https://spark.apache.org/.

ORAAH's architecture and approach to big data analytics is to leverage the cluster compute infrastructure for parallel, distributed computation. ORAAH does this while shielding the R user from Hadoop's complexity through a small number of easy-to-use APIs.

2 What's new in ORAAH release 2.7.1

1. Support for Cloudera Distribution of Hadoop (CDH) release 5.11.0. Both "classic" MR1 and YARN MR2 APIs are supported.
2. ORAAH 2.7.1 extends support for Apache Spark. Select predictive analytic functions can be executed on a Hadoop cluster using YARN to dynamically form a Spark cluster on a dedicated standalone Spark cluster. Users can switch Spark execution on or off using new `spark.connect()` and `spark.disconnect()` functions. For more information about Spark, refer to https://spark.apache.org/.
3. In addition to Distributed Model Matrix (DMM) that can be saved in Comma-Separated Values (CSV) format on HDFS via a call to `hdfs.write()`, in release 2.7.1 `hdfs.write()` now adds support for writing Spark DataFrame objects as well.
4. ORAAH 2.7.1 is fully compatible with OAAgraph package and introduces tight integration with Oracle's Parallel Graph AnalytiX (PGX) in-memnory database. Supported PGX version 2.4.1 and up. See http://www.oracle.com/technetwork/-oracle-labs/parallel-graph-analytix for more information.
5. ORAAH 2.7.1 can now connect to Oracle Databases running in Multitenant Container Database (CDB) mode. A new parameter *pdb* can be used to specify the service name of the Pluggable database (PDB) to which the connection has to be established.
6. The performance of `ore.create()` for Hive has been improved significantly in this release. Also, users can create a table in another Hive database instead of the one they are connected to, if needed. For example, if user is connected to *default* database and wants to create a table *sample* in *hivedb1* database, the ore.create *table* parameter should be "hivedb1.sample". Also a new parameter *append* for `ore.create` lets you append an `ore.frame` or `data.frame` to an existing Hive table specified by the *table* parameter. If the table does not exist, and `append=TRUE`, the warning is reported and a new table is created.

7. ORAAH 2.7.1 now supports a new environment configuration variable `ORCH_CLASSPATH`. You can set the `CLASSPATH` used by ORAAH using this variable. `ORCH_CLASSPATH` can be set for the shell or in *Renviron.site* file to make it available for R and ORAAH. *rJava* does not support wildcard characters in `CLASSPATH` environment variable. This led to users adding exact jar files in the `CLASSPATH` before they could use Apache Spark analytics. Now ORAAH 2.7.1 resolves this issue and will support the wildcards in path. For example, having a path */usr/lib/hadoop/lib/*.jar* in `CLASSPATH` adds all jars from */usr/lib/hadoop/lib* to rJava CLASSPATH. This makes the process of getting started with ORAAH's Spark features easier and faster. You can use wildcards with ORCH_CLASSPATH too, if you are going to use `ORCH_CLASSPATH` instead of `CLASSPATH` for ORAAH configuration.

8. This version adds new features to distributed model matrix and distributed formula and improves its performance and support in all ORAAH and Spark MLlib-based functionality of ORAAH. See "Change List" document for more details.

9. Note that the new MKL is added to the installer package: Intel® Math Kernel Library Version 2017 for Intel® 64 architecture applications.

10. This version of ORAAH comes with improved installers and un-installers for both ORAAH server and client. The installer now will check your environemnt and run a set of validation checks to make sure all pre-requisits are available and minimal requirements for running ORAAH are met. Uninstallation procedure has been improved as well and will work better with co-existing Oracle R Enterprise and 3rd party installed packages. See "Install Guide" and "Change List" document for more details on improvements and new features.

11. Bug fixes and updates across the platform improve stability and ease-of-use. For more information see the "Change List" document.

3 What's new in ORAAH since release 2.7.0

1. Since 2.7.0 version of ORAAH we have introduced support for selected algorithms from Apache MLlib with support of our proprietary optimized data transformation and data storage algorithms built on top of Apache Spark and designed to improve performance and interoperability of MLlib and ORAAH machine learning functionality with R.

2. ORAAH 2.7.0 has introduced distributed model matrix and distributed formula that are used in all Spark MLlib-based functionality to greaty improve performance and enhance functional compatibility with R. Machine learning and statistical algorithms require a Distributed Model Matrix (DMM) for their training phase. For supervised learning algorithms DMM captures a target variable and explanatory terms; for unsupervised learning DMM captures explanatory terms only. Distributed Model Matrices has its own implementation of R formula, which closely follows R formula syntax, and much more efficient from performance perspective. Internally Distributed Model Matrices are stored as Spark RDDs (Resilient Distributed Datasets).

3. New functions orch.save.model and orch.load.model were added that allows to save and load models created using Apache Spark MLlib in ORAAH to HDFS for scoring/prediction later on. It also enables model sharing amongst different users if the other users have access to the path where models are saved.

4. Since 2.7.0 release Distributed Model Matrix (DMM) can be saved in Comma-Separated Values (CSV) format on HDFS via a call to hdfs.write(). For instance, the below example solves a linear regression model, generates predictions, and saves them on HDFS in CSV format.

5. Since 2.7.0 release ORAAH supports HiveServer2 in the Hive transparency layer. ORAAH uses JDBC/Thrift connection to communicate with Hive and execute Hive queries on a remote or local Hadoop cluster. This overhaul of connectivity to Hive replaces the error-prone and unstable CLI execution layer used in previous versions of ORAAH. You will find great improvement in the performance of the Hive transparency layer as well as lower latency in Hive queries. These changes provide a fluent and robust integration of ORAAH/R with Apache Hive. For more information about Apache Hive refer to https://hive.apache.org/.

6. R's base predict() function was extended to support prediction on Spark when a model generated by GLM on Spark or by MLlib-base analytics functions is used.

7. Since 2.7.0 release new implementations on Spark for GLM, LM and Neural networks were added. Because these new functions are not compatible with the currently available functions on Hadoop API, they are implemented as orch.glm2(), orch.lm2() and orch.neural2(), separate functions that can only be executed when ORAAH is connected to Spark.

8. Additinal options to control JVM -Xmx and XX:MaxPermSize options were added in ORAAH 2.7.0 - `ORCH_JAVA_XMX` and `ORCH_JAVA_MAX_PERM`. Both are exposed as environment variables and can be set in user's shell before starting ORAAH in order to force ORAAH to use non-default Java options.
9. Since ORAAH 2.5.0 release support for data load to Spark cache from HDFS files was introduced. This is provided by the function hdfs.toRDD(). ORAAH dfs.id objects were also extended to support data residing in both HDFS and in Spark memory.
10. Neural network analytical function (orch.neural) now supports execution on Spark in addition to previously available Hadoop mapReduce execution support, which dramatically improves its performance. There are no changes to the neural network APIs, its functionality transparently switched to Spark execution facilities once ORAAH is connected to Spark (see spar.connect()). Details are available in orch.neural() documentation.

4 Known issues in ORAAH release 2.7.1

There are a few known issues in this release.

1. Attaching an HDFS folder that contains only 0-sized data files results in an exception. To work around this, prior to using the hdfs.attach() API on such files, use the hdfs.size() function to check if the size is 0.
2. Executing a MapReduce job that produces no output at all may result in exception that stops script execution. Set the parameter cleanup=TRUE to avoid this exception.
3. Ensure all categorical columns are explicitly converted into factors (as.factor(categorical column)) in the formula before using such columns in orch.neural().
4. Working with Spark requires that all necessary Spark and Hadoop Java libraries are included in CLASSPATH environment variable. Otherwise, connections to Spark will fail. See the installation guide for detailed information about configuring ORAAH and the system environment.
5. Only text-based HDFS objects can be loaded into Spark distributed memory and used for Spark-based analytics. Binary RData HDFS objects are not supported in Spark.
6. The `hdfs.fromHive` function still uses CLI to interact with the Hive metastore. This may cause issues if the Hive client is not installed or is misconfigured on the client host.

5 Parallelization Packages in R Compared to Those in ORAAH

ORAAH is often compared/contrasted with other options available in the R ecosystem, in particular, with the popular open source R package called "parallel". The parallel package provides a low-level infrastructure for "coarse-grained" distributed and parallel computation. While it is fairly general, it tends to encourage an approach that is based on using the aggregate RAM in the cluster as opposed to using the file system. Specifically, it lacks data and resource management components, a task scheduling component, and an administrative interface for monitoring. Programming, however, follows the broad MapReduce paradigm.

The crux of ORAAH is read parallelization and robust methods over parallelized data. Efficient parallelization of reads is the single most important step necessary for big data analytics because it is either expensive or impractical to load all available data into memory addressable by a single process. The rest of this document explains the architecture, design principles and certain implementation details of the technology.

6 Architecture

ORAAH is built upon the Hadoop streaming utility. Hadoop streaming enables creation and execution of MapReduce jobs with any executable or script as mapper and/or reducer. ORAAH automatically builds the logic required to transform an input stream of data into an R data frame object that can be readily consumed by user-provided snippets of mapper and reducer logic written in R.

ORAAH is designed for R users to work with Hadoop cluster in a client-server configuration. Client configurations must conform to the requirements of the Hadoop distribution that ORAAH is deployed against. This is because ORAAH uses command line interfaces to communicate from the client node to HDFS and HIVE running on Hadoop cluster.

ORAAH allows R users to move data from an Oracle Database table/view into HDFS files in Hadoop. To do this, ORAAH uses the SQOOP utility. Similarly, data can be moved back from an HDFS file into Oracle Database. For the reverse data movement, a choice of using SQOOP or the Oracle Loader for Hadoop (OLH) utility is available and the choice depends on the size of data being moved and security requirements for connecting to Oracle Database.

For performance-sensitive analytic workloads, ORAAH supports R's binary RData representation for both input and output. Conversion utilities from delimiter-separated representation of input data to/from RData representation is available as part of ORAAH.

ORAAH includes a Hadoop Abstraction Layer (HAL) which manages the similarities and differences across various Hadoop distributions. ORCH will auto-detect the Hadoop version at startup but to use ORAAH on a different distribution see the help regarding ORCH_HAL_VERSION for details.

ORAAH consists of five distinct sets of APIs with the following functionality:

1. Access to HDFS objects: ORCH HDFS APIs map 1:1 to the corresponding HDFS command line interfaces that explore HDFS files. Additionally, ORAAH exposes new functions to explore contents of HDFS files.
2. Access to HIVE SQL functionality by overloading certain R operations and functions on data frame objects that map to HIVE tables/views.
3. Launching Hadoop jobs from within a user's R session using map/reduce functions written in R language. Such functions can be tested locally (help(orch.dryrun) and help(orch.debug()). ORAAH mimics Hadoop style computation but on the client's R environment optionally enabling debugging capability.
4. Native, Hadoop-optimized analytics functions for commonly used statistical and predictive techniques.
5. File movement between local user's R environment (in-memory R objects), local file system on a user's client machine, HDFS/HIVE and Oracle Database. ORAAH can work with the following types of input:
   - Delimited text files that are stored in a HDFS directory. (ORAAH recognizes only HDFS directory names as input and processes all files inside the directory.
   - Apache HIVE tables.
   - Proprietary binary RData representations.
   - Apache Spark RDD objects.

If data that is input to an ORAAH-orchestrated MapReduce computation does not reside in HDFS (for example, a logical HiveQL query), ORAAH will create a copy of the data in HDFS automatically prior to launching the computation.

Before ORAAH works with delimited text files in a HDFS directory, it determines the metadata associated with the files and captures this metadata in a separate file named *ORCHMETA*, which is stored alongside the data files.

ORCHMETA contains the following information:

a. whether the file contains key(s) and if so the delimiter that is the key separator;
b. the delimiter that is the value separator;
c. the number and data types of columns in the file;
d. optional names of columns (which are available in a header line);
e. dictionary information for categorical columns (in R terminology, the levels of a factor column);
f. other ORAAH-specific system data.

ORAAH runs an automatic metadata discovery procedure on HDFS files as part of `hdfs.attach()` invocation. When working with HIVE tables, the *ORCHMETA* file is auto-created from the HIVE table definition.

ORAAH can optionally convert input data into R's binary RData representation (See the documentation for `hdfs.toRData` and `hdfs.fromRData`). RData representation is recommended because I/O performance of R-based MapReduce jobs on such representation is on par with a pure Java based MapReduce implementation.

ORAAH captures row streams from HDFS files and delivers them formatted as a data frame object (or optionally matrix/vector/list objects generated from the data frame object or "as is" if RData representation is used) to mapper logic written in R. To accomplish this, ORAAH must recognize the tokens and data types of the tokens that become columns of a data frame. ORAAH uses R's facilities to parse and interpret tokens in input row streams. If missing values are not represented using R's "NA" token, they can be explicitly identified by the *na.strings* argument of `hdfs.attach()`.

Delimited text files with the same key and value separator are preferred over files with varying key delimiter and value delimiter. Read performance of files with the same key and value delimiter is roughly two times better than that of files with different such delimiters. The key delimiter and the value delimiter can be specified through the *key.sep* and *val.sep* arguments of `hdfs.attach()`, or when running a MapReduce job for its output HDFS data.

IMPORTANT: Binary RData representation is the most performance efficient representation of input data in ORAAH. When possible, it is suggested to use this binary data representation for performance sensitive analytics.

As of release 2.5.0, ORAAH has included support for Spark Resilient Distributed Dataset (RDD) objects. Users can load data from HDFS to Spark's RDD and optionally cache it in Spark distributed memory in order to improve performance of Spark-enabled analytics run on this data set. Results of Spark-enabled analytics are retuned as another Spark RDD object. Users will have an option to save data from Spark's distributed memory to an HDFS directory in order to preserve it between ORAAH sessions. For more information, review the documentation of the `hdfs.toRDD()` and `hdfs.write()` functions.

7 Working With the ORAAH Hadoop Interface

ORAAH's MapReduce job definition is accomplished through `hadoop.exec()` or `hadoop.run()` APIs. The `hadoop.exec()` function is a subset of `hadoop.run()` in that `hadoop.run()` additionally supports appropriate staging of the final result from a MapReduce job. Use the `hadoop.exec()` function to build a pipeline of ORCH MapReduce job that stream data from one job stage to another when all of the following is true:

- The job is not required to bring intermediate results back to ORAAH environment.
- The input to the MapReduce job is already in HDFS.

See help("hadoop.exec") and help("hadoop.run") for more details.

By default, ORAAH prepares the rows streamed into the mappers and reducers defined as part of hadoop.run() or hadoop.exec() APIs as a data.frame object with input values and a vector object with input keys. Optionally, input can be supplied to the mapper as an R matrix, vector, or a list object (see mapred.config()). If the input data to a MapReduce job is in RData representation, set the "direct.call" parameter to TRUE. This will cause ORAAH to pass input data "as is" to the user-supplied R mapper code. The R code that is meant to execute as mapper logic requires no special coding. It is nothing more than an R closure that accepts a data.frame object as input (by default) together with other variables exported from the user's client R session. Optionally, several configuration parameters may be specified by instantiating an R object of "mapred.config" class.

The mapper has a choice of generating either structured output in the form of a data.frame or any arbitrary data structure. By default, the mapper is assumed to perform any filtering/computation on data it receives as input and generate as output a data frame with the same number of columns and same data type as its input. However this can be overridden through defining the structure of the mapper's output and hence the reducer's input (in a map/reduce sequence) object using "map.output" parameter in the "mapred.config" class. Additionally, if the mapper's output is 'pristine' (i.e. all null values are represented either as "NA" or "") then, optionally setting yet another "mapred.config" parameter called mapred.pristine can enable better read performance in the reducer. See orch.keyvals() and orch.keyval() for additional details.

Complex output objects from the mapper can be packed using orch.pack() (and read using corresponding orch.unpack()) interfaces before being passed to orch.keyval[s]() functions. Packing of objects provides the benefit of allowing a user to store unstructured, semi-structured or variable-structured data in a structured output of MapReduce jobs. Also, packed data can be

compressed in order to minimize space and improve performance. See the "COMPRESS" argument description of the orch.pack() function.

By default, the reducer is not assumed to generate output data in the same format as its input. The user can specify the structure of the reducer's output via the "reduce.output" field of "mapred.config" class. If this specification is unavailable, ORAAH can discover the metadata by sampling the reducer's output.

If the mapper task is computationally expensive and likely to take longer than the default Hadoop timeout of 600 seconds, then the "task.timeout" parameter of "mapred.config" class must be set to allow longer timeouts for a specific task. The timeout value must be specified in seconds.

8 Working with ORAAH Hive Interface

ORAAH allows Hive tables/views from default and non-default databases to be used as 'special' data frames in that they have enough metadata to be able to generate HiveQL queries when data in the Hive tables/views is processed. See the following demos to understand ORAAH-Hive interfaces:

```
R> demo(package="ORCH")
hive_aggregate Aggregation in HIVE
hive_analysis Basic analysis & data processing operations
hive_basic Basic connectivity to HIVE storage
hive_binning Binning logic
hive_columnfns Column function
hive_nulls Handling of NULL in SQL vs. NA in R
hive_pushpull HIVE <-> R data transfer
hive_sequencefile Creating and using HIVE table
```

9 Native Analytic Functions

The following analytic functions are available out-of-the-box in ORAAH. These functions are parallel and distributed, and execute utilizing all nodes of Hadoop cluster.

1. Covariance matrix computation (`orch.cov`)
2. Correlation matrix computation (`orch.cor`)
3. Principal Component Analysis (`orch.princomp`, `orch.predict`)
4. K-means clustering (`orch.kmeans`, `orch.predict`)
5. Linear regression (`orch.lm`, `orch.predict`)
6. Generalized linear models including logistic regression (`orch.glm`)
7. Neural Networks (`orch.neural`)
8. Matrix completion using low rank matrix factorization (`orch.lmf`)
9. Non negative matrix factorization (`orch.nmf`)
10. Reservoir sampling (`orch.sample`)

10 Copyright Notice